Programming assignment #1

Multicore Systems
Game of Life

• John Conway’s game of life
  – Cellular automation

• Universe
  – Infinite 2D orthogonal grid of square cells
  – Cell is one of two states: LIVE or DEAD
  – Each time step, cell lives or dies depending on 8 neighbor cells

• Rules
  – Any live cell with \#live_neighbors < 2 or \#live_neighbors > 3 will die
  – Any dead cell with 2 \leq \#live_neighbors \leq 3, will come to live
3D Game of Life

- Cubic cells in 3D universe
  - One cell will have $26(3^3 - 1)$ neighbors

- Rules
  - $N = \# \text{ of live neighbors}$
  - If a LIVE cell, with $N < \text{dead_min}$ or $N > \text{dead_max}$, will die
  - If a DEAD cell, with $\text{live_min} < N < \text{live_max}$, will come to live

- Example
  - dead_min=5, dead_max=20, live_min=10, live_max=15
Additional Rule

- Devil
  - Devil moves each step randomly
  - Cell state occupied by devil comes to PLAGUE state
  - Plague cell toggles neighbor cells (Live → Dead / Dead → Live)
  - If devil count comes to zero, one devil revives at random cell
  - Devil number doubles at each step
  - Initial devil number is one
Additional Rule

• Angel
  – Angel moves proportional value to map size
  – Angel moves each step to direction at largest number of devil
  – Angel scope value is equal to devil number
  – If some devils place in angel scope, they are removed at game
  – If some PLAGUE cells are in angel scope, they restore previous state
  – Angel number is just one
Programming

• 3D Game of Life in two programming models
  – Sequential programming
  – Parallel programming with Pthreads, OpenMP, MPI

• Input file
  – Parameters about DEAD, LIVE, map size, steps to run
  – Random seed for map generation and devil move/revival

• Output
  – Measured time
  – Output file
    ▪ Result universe of Game of Life first/last map
    ▪ Result of Angel/Devils first/last position
Initial attribute

• Map
  – Size of map
  – Random seed for initiating map

• Dead / Live
  – Minimum and Maximum value

• Devil
  – Random seed for devil move and revival
Input example (1/2)

- input.life
  - total_steps: 100
  - map_size: 10
  - dead_min: 5
  - dead_max: 20
  - live_min: 10
  - live_max: 15
• input.life
  – SEED_MAP_INIT: 1
  – SEED_DVL_GEN_X: 2
  – SEED_DVL_GEN_Y: 3
  – SEED_DVL_GEN_Z: 4
  – SEED_DVL_MOV_X: 5
  – SEED_DVL_MOV_Y: 6
  – SEED_DVL_MOV_Z: 7
Output

- Measured time

- Initial_map.txt / Final_map.txt
  - At start/finish game, make initial/final cell status

- Initial_pos.txt / Final_pos.txt
  - At start/finish game, make angel/devil position

- Cell status
  - Dead: D
  - Live: L
  - Plague: P
Output example

- **Initial_map.txt / Final_map.txt**
  - 3 x 3 x 3 universe input

```
$ cat Initial_map.txt
D L D
D L D
D L D
D D L
D D D
D D L
D D P
L D P
```

```
X
0
0
1
1
2
2
```
Output example

- Initial_pos.txt / Final_pos.txt
  - 5 x 5 x 5 universe input

```
$ cat Initial_pos.txt
[Angel]
(2, 2, 2)

[Devil]
(0, 0, 0)
```
Flow chart

Measurement
total overhead

Game-start

Devil stage
count++

LIVE & DEAD stage
count++

PLAGUE stage
count++

Angel stage
count > iteration?

Game-end

No
Yes
Initialize map

- lcgrand.c / lcgrand.h
  - Random value generator
  - `uniform(lower, upper, RANDOM_SEED)`
  - `uniform()` function will return $X$
    - lower $\leq X \leq$ upper
  - `uniform()` function will return different value about different RANDOM_SEED
  - However, `uniform()` function will return SAME SEQUENCE of value about same RANDOM_SEED
    - For example, when use seed 1, always `uniform()` return 1, 100, 54, 7, …
Initialize map

• **Sequence**
  1) Fill total cell state to LIVE or DEAD by using `uniform()`
     - `uniform(0, 1, SEED)` will return 0 or 1
     - 0 is dead / 1 is live
  2) Position one Angel to \( P = \text{ROUND}(\text{map\_size}/2) -1, (P, P, P) \)
  3) Position one Devil by using `uniform()`
  4) Generate initial position of angel & devil
  5) Generate initial state of cell
Devil stage

- If devil count == 0
  - Generate one devil
- Else
  - Move devils by using x, y, z \textit{uniform}(0, 2, \textit{RANDOM\_SEED})
  - 0: +direction mov, 1: hold position for direction, 2: -direction mov
  - If that direction move is over map range, hold position about that direction
  - If one more devils are in same cell, only one devil survives and others are removed

- After that, devils change cell state to PLAGUE

- Next, devils copy them and copies are positioned by using \textit{uniform}() 
  - Example: 2 devils \(\rightarrow\) 4 devils / 8 devils \(\rightarrow\) 16 devils
LIVE & DEAD stage

- LIVE & DEAD cells change state following Game of Life rule
- Only consider about LIVE & DEAD cells
PLAGUE stage

• PLAGUE state cells toggle their neighbor cells

• Example

```
  L D L D L
  D L D L D
  L D P D L
  D L D L D
  L D L D L
```

```
  L D L D L
  D D L D D
  L P L L P
  D D L D D
  L D L D L
```
Angel stage

- Angel scope is set to current devil count

- Angel is moved to direction of current largest devil count
  - Direction priority: +x, +y, +z, -x, -y, -z
  - Angel move value is proportional to map size (map_size / 10)
  - Each stage, angel can move just one direction
  - If angel is over map range, angel’s position is edge of map
    - In 30 x 30 x 30 universe, if angle’s position is (2, 2, 2) and direction is –x, then angel’s position is (0, 2, 2)

- PLAGUE state cells in angel scope restore previous state

- Devils in angel scope are removed at game
Source code

- Posted at CSL assignment

- Given source code list
  - life.c
  - setup.h
  - game.c
  - game.h
  - list.c
  - list.h
  - lcgrand.c
  - lcgrand.h
  - example input.life
  - Makefile
Submission

• Submit through iCampus

• Due date
  – Sequential project: 4/10 ~ 4/20
  – Pthread project: 4/21 ~ 4/30
  – OpenMP (exercise & project): 5/1 ~ 5/7 & 5/8 ~ 5/18

• StudentID.tar
  – Makefile
  – game.h
  – game.c
  – README (not mandatory)
Additional items

• T.A.: Jaehyun Song (Semi-building #400509)
  – jaehyun.song@csl.skku.edu

• Penalty
  – Delayed submission: -10% per day

• Score
  – Total project/example score of multicore systems: 30% / 10%
  – Right answer: 50%
  – Speed up = my parallel time / fastest seq time
  – Your score = (Your speed up / MAX speed up) * 50%
  – Sequential, Pthread, OpenMP, MPI = (TBD : 1 : 1 : 1)